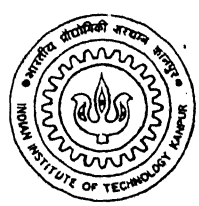
Higher Order Visibly Pushdown Languages

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Shaji Illias C K



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

June, 2005

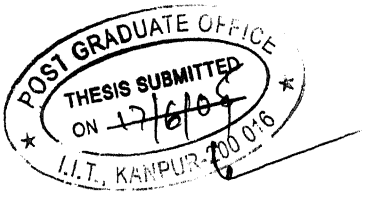
Certificate

This is to certify that the work contained in the thesis entitled " *Higher Order Visibly Pushdown Languages* ", by *Shaji Illias C K*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

June, 2005

Anil Seth

(Dr.Anil Seth)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.



Abstract

Visibly pushdown automata (VPA) has been recently defined as a restricted version of pushdown automata. It has been shown that the class of languages recognized by them (VPLs) are closed under all the boolean operations and have tractable decision problems.

We introduce a new variation of higher pushdown automata named "higher order visibly pushdown automata" (HVPAs) as an extension to VPAs and the class of languages accepted by them (HVPLs) is studied. In the higher order pushdown automata the symbols appearing on pushdown stacks are no longer ordinary symbols but stacks themselves. We show that the set of languages accepted by this model is closed under union, intersection, renaming operation etc. We also prove that HVPLs are not closed under determinization, concatenation and Kleene * by giving counter examples.

We also look at the omega language accepted by HVPAs (ω-HVPLs) and shows that ω-HVPAs with Buchi and Muller acceptance conditions are equivalent.

Acknowledgements

I take this immense opportunity to express my sincere gratitude toward my supervisor Dr.Anil Seth for his invaluable guidance and suggestions which made my thesis successful. His patience during discussions, encouragement of new ideas was always a great motivation for me during my thesis. I consider myself extremely fortunate to have had a chance to work under his supervision. It has been a very enlightening and enjoyable experience to work under him and I express heart-felt thanks to him.

I also wish thank whole heartily all the faculty members of the Department of Computer Science and Engineering for the invaluable knowledge they have imparted to me and for teaching the principles in most exciting and enjoyable way. I also extend my thanks to the technical and office staff of the department for maintaining an excellent work facility.

I am also thankful to our classmates for all the cheerful support they gave to me. I will also like to thank Tonychayan, Ramachandran, Prince and Sumesh for making my stay at IITK so enjoyable, and cheerful. I would also like to thank my parents for encouraging me all the time and letting me to take my own decisions.

Contents

Chapter 1

Introduction

Recently in [1] the class of visibly pushdown languages (VPLs) has been defined as a subclass of context free languages. It has been shown that VPLs are closed under all boolean operations and determinizable (and hence closed under complementation). And decision problems viz emptiness, inclusion and universality are shown to be decidable. In VPLs the input alphabet is partitioned into three sets namely call, return and internal symbols. The operation on the stack is determined by the type of the alphabet read by the automata. More precisely a push (pop) operation is always performed when an alphabet of type call (return) is read. When the automaton reads an internal symbol it can change only its state by leaving the stack contents unaltered.

VPLs are relevant to several applications that use context free languages such as model checking of software programs and their pushdown models. In [3], Pitcher suggests VPLs to be the best model for semantics of effects in processing XML streams. Games on visibly pushdown systems are also been defined and its topological complexity is studied in [4] and [5].

Higher order pushdown automata (HPDA) was first considered by Aho and Ullman in [6]. They showed that 2-level pushdown automata recognize the indexed languages. In [7] and [8] Maslove shows that k-level pushdown automata correspond to k-level indexed grammars. In [9] Damm and Geordt gives a characterization of

k-level pushdown automata in terms of k-level OI-indexed grammars. In general, the hierarchy of languages accepted by different level higher order pushdown automata are viewed as an infinite Chomsky hierarchy, called OI-hierarchy with finite automaton at the 0th level, context free languages on the first level and so on.

In higher order pushdown automata the symbols appearing on pushdown stacks are no longer ordinary symbols but stacks themselves. These kind of stacks are called higher order stores. Stores of *level-1* are sequences of symbols from some finite set of alphabets. In fact *level-1* pushdown store is the ordinary stack. Stores of *level-n* are sequences of *level-(n-1)* stores. The allowed operations on this store are,

1. ordinary push and pop operations on top *level-1* stack

2. *k-order* push and pop operations allowing to duplicate or erase the top-most level *level-k* store of any given level $k \leq n$.

It was always been a matter of interest to look at some restricted model of HPDAs which have more closure properties than ordinary HPDAs have. Here we are introducing a restricted version of HPDAs named "higher order visibly pushdown automata" (HVPA) and the language accepted by it (HVPL) is studied. Similar to VPAs in the case of HPVAs also we partition the input alphabets into three classes namely call, return and internal. For an *n-level* HVPA the call and return sets are further partitioned to n sets say $call_1$,...$call_n$ and $return_1$,...$return_n$. While reading the input alphabet, the behavior (push and pop) of automata is decided by in which partition the input alphabet read belongs. i.e, for alphabets in the same class the stack of automata behaves in the same manner. When the automaton reads a $call_i$ ($return_i$) synibol then it performs a $push_i(pop_i)$ operation. When it reads an internal symbol, as in case VPAs it changes the state and leaves the stack unaltered. This restriction makes it easier to prove some of the closure properties.

In this work we prove that HVPLs are closed under union, intersection, renaming operation etc. We also prove that HVPLs are not closed under determinization, concatenation and Kleene * by giving counter examples.

We also look at the omega language accepted by HVPAs and shows that the Buchi and Muller acceptance conditions are equivalent. Further we show that, ω-HVPL are not determinizable.

Chapter 2

Definitions

2.1 Visibly Pushdown Automata(VPA)

A pushdown alphabet is a tuple $\overline{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ that comprises three disjoint finite alphabets- Σ_c is a finite set of calls, Σ_r is a finite set of returns and Σ_{int} is a finite set of internal actions. For any such $\overline{\Sigma}$, let $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$

We define pushdown automata over $\overline{\Sigma}$. Intuitively, the pushdown automaton is restricted such that it pushes into the stack only when it reads a call, it pops the stack only at returns, and does not use the stack when it reads internal actions symbols. The input hence controls the kind of operations permissible on the stack - however there is no restriction on the symbols that can be pushed or popped. We call such an automaton a *visibly pushdown automaton*, defined as follows:

Definition (Visibly Pushdown Automata) *A visibly pushdown automaton on finite words over* $\langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ *is a tuple* $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ *where* Q *is a finite set of states ,* $Q_{in} \subseteq Q$ *is a set of initial symbols,* Γ *is a finite set of stack alphabets that contains a special bottom-of-stack symbol* $\perp$ *(a symbol which is never popped),* $\delta \subseteq (Q \times \Sigma_c \times \Gamma \times Q \times (\Gamma \setminus \{\perp\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{int} \times \Gamma \times Q)$*, and* $Q_F \subseteq Q$ *is a set of final states.*

A transition (q,a,γ',q',γ) where $a \in \Sigma_c$ and $\gamma \neq \perp$, is a push-transition where on reading a, γ is pushed onto the stack and the control changes from state q *to* q'. Similarly (q,a,γ,q') is a pop-transition where γ ($\gamma \neq \perp$) is read from the top of the stack and popped, and the control state changes from q to q'. If the top symbol on stack is $\perp$, machine does not pop it, it just changes the control state from q to q'. Note that on internal actions there is no stack operation.

A stack is a nonempty finite sequence of over Γ ending in the bottom-of-stack symbol $\perp$; let us denote the set of all stacks as $St = (\Gamma \setminus \{\perp\})^*.\{\perp\}$. For a word $w = a_1 \dots a_k$ in Σ^*, a run of M on w is a sequence $\rho = (q_0, \sigma_0), \dots, (q_k, \sigma_k)$, where $q_i \in Q$, $\sigma_i \in St$, $q_0 \in Q_{in}$, $\sigma_0 = \perp$ and for every $i \in [1,k]$ the following holds:

[Push] If a_i is a call, then $\exists \gamma \in \Gamma$ such that $(q_i, a_i, \gamma', q_{i+1}, \gamma_i) \in \delta$ and $\sigma_{i+1} = \gamma.\sigma_i$.

[Pop] If a_i is a return, then $\exists \gamma \in \Gamma$ such that $(q_i, a_i, \gamma, q_{i+1}) \in \delta$ and either $\gamma \neq \perp$ and $\sigma_i = \gamma.\sigma_{i+1}.$, or $\gamma = \perp$ and $\sigma_i = \sigma_{i+1} = \perp$.

[Internal] If a_i is an internal action, then $(q_i, a_i, \gamma, q_{i+1}) \in \delta$ and $\sigma_{i+1} = \sigma_i$.

A run $\rho = (q_0, \sigma_0), \dots, (q_k, \sigma_k)$ is accepting if the last state is a final state, i.e if $q_k \in Q_F$. A word $w \in \Sigma^*$ is accepted by a VPA M if there is an accepting run of M on w. The language L of M, is the set of words accepted by M and a language is said to be *visibly pushdown language* (VPL) if there is some VPA to accept it.

2.2 Higher Order Pushdown Automata(HPDA)

We define *level n store(stack)* in a recursive manner. A *level 1 store*(or *1-store*) over a finite set of stack alphabet Γ is an arbitrary sequence of $[\gamma_1, \dots, \gamma_l]$ of Γ with $l \geq 0$. A *level n store* or *n-store*, for $n \geq 2$, is a sequence $[s_1, \dots, s_l]$ of (n-1) stores where $l \geq 0$. Let Q be the set of states of the PDA and Σ be the set of input alphabets. Then following operations can be performed on *1-store*: for $\gamma \in \Gamma$, $a \in \Sigma$ and q_i, $q_j \in Q$

$push_1^{\gamma}(q_i, a, \gamma_l, [\gamma_1, \ldots, , \gamma_{l-1}, \gamma_l]) := (q_j, [\gamma_1, \ldots, \gamma_{l-1}, \gamma_l, \gamma])$

$pop_1(q_i, a, \gamma_l, [\gamma_1, \ldots, , \gamma_{l-1}, \gamma_l]) := (q_j, [\gamma_1, \ldots, , \gamma_{l-1}])$

If $[s_1, \ldots, s_{l-1}, s_l]$ is a store of level $n > 1$ and let γ be the stack top of topmost *1-store* of s_l, then the following operations are possible:

$push_n(q_i, a, \gamma, [s_1, \ldots, s_{l-1}, s_l]) := (q_i, [s_1, \ldots, s_{l-1}, s_l, s_l])$

$push_k(q_i, a, \gamma, [s_1, \ldots, s_{l-1}, s_l]) := (q_j, [s_1, \ldots, s_{l-1}, push_k(s_l)])$ if $2 \leq k < n$

$push_1^{\gamma}(q_i, a, \gamma, [s_1, \ldots, s_{l-1}, s_l]) := (q_j, [s_1, \ldots, s_{l-1}, push_1^{\gamma}(s_l)])$

$pop_n(q_i, a, \gamma, [s_1, \ldots, s_{l-1}, s_l]) := (q_j, [s_1, \ldots, s_{l-1}])$

$pop_k(q_i, a, \gamma, [s_1, \ldots, s_{l-1}, s_l]) := (q_j, [s_1, \ldots, s_{l-1}, pop_k(s_l)])$ if $1 \leq k < n$

Note that *level 1* operations are ordinary push and pop operations.

Given Γ, Σ and n, the set Op_n of operations (on a store) of level n consists of : $push_k$ for $2 \leq k \leq n$, $push_1^{\gamma}$ for $\gamma \in \Gamma$, and pop_k for $1 \leq k \leq n$.

Definition (Higher order Pushdown automata) *A higher order Pushdown automata of level n (or n-HPDA) over a finite set of input alphabets Σ is a tuple $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ where Q is a finite set of states, $Q_{in} \subseteq Q$ is a set of initial states, $Q_F \subseteq Q$ is a set of final states, Γ is a finite set of stack alphabets and $\delta \subseteq (Q \times \Sigma \times \Gamma \times Q \times Op_n)$ is the transition function.*

In our definition of HPDA we do not allow any ϵ-moves.

Language of an HPDA is defined in a common way. Set of languages accepted by *level n* HPDA is called *level n languages.*

2.3 Higher Order Visibly Pushdown Automata(HVPA

In this section we introduce a new model of computational called *Higher Order Visibly Pushdown Automata(HVPA).* Basically it is a VPA with an iterated stack

(*level n-stote*). As in the case of ordinary VPA, the kind of operations permissible on the stack of HVPA is controlled by the input symbol read.

Let $\overline{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ be a set of pushdown alphabet. Here we have a partition on both Σ_c and Σ_r. i.e $\Sigma_c = \Sigma_{c_1} \cup \ldots \cup \Sigma_{c_n}$ and Σ_{c_i}'s are pairwise disjoint (i.e $\Sigma_{c_i} \cap \Sigma_{c_j} = \phi$ for $i \neq j$). Similarly $\Sigma_r = \Sigma_{r_1} \cup \ldots \cup \Sigma_{r_n}$ and Σ_{r_i}'s are pairwise disjoint (i.e $\Sigma_{r_i} \cap \Sigma_{r_j} = \phi$ for $i \neq j$). The partition on Σ_c (Σ_r) determines which order push(pop) operation is performed by the machine. That is, if the machine reads a symbol in Σ_{c_i} (Σ_{r_i}), a $Push_i$ (Pop_i) is performed on the stack. One interesting thing to notice is that, non-determinism does not have much role to play when we perform a $Push_i$ or Pop_i for $i > 1$, because the stack would be exactly the same in every possible moves. So when the machine reads an alphabet which belongs to Σ_{c_i} (Σ_{r_i}) for $i > 1$, non-determinism is restricted to making choice among various possible states only.

Assumptions about the model: When we start the computation, we assume that we have an empty *n store* with the special bottom-of-stack symbol $\perp$ on its *1-store*. As in case of VPAs we make sure that $\perp$ symbol is never popped. Also ϵ-moves are not allowed in HVPAs so that the machine is forced to read an input symbol in every move.

Another assumption we make is that, the machine does not pop the last *(i-1)-store* from an *i-store*. Instead of that the machine signals that the *store* we are trying to pop is the last *store* in the immediately enclosing *store*. Depending on this information the machine can decide to which state it should move. In a way this is a natural extension of the assumption that $\perp$ is never popped from a *1-store*. In order to make this invariant possible, we assume that the machine queries every time before it makes a transition and the answer is used in making the decision. Let σ denotes the *store* of the HVPA, we define a function *Is_Last*, which takes σ and the input alphabet currently reading as its parameters. *Is_Last* is defined as follows :

$Is\_Last(\sigma, a)$ returns 'Y' if $a \in \Sigma_{r_i}$ and the stack going to be popped is the last stack in the topmost *(i+1)-store* and 'N' otherwise. We make this invariant as part of the transition relation. Now we define HVPA formally as follows:

Definition (Higher Order Visibly Pushdown Automata(HVPA)) *A higher order (of level n) VPA on finite words over* $\overline{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ *is a tuple* $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ *where* Q *is a finite set of states,* $Q_{in} \subseteq Q$ *is the set of initial states,* $Q_F \subseteq Q$ *is the set of final states* Γ *is the finite set of stack alphabets that contains a special bottom-of-stack symbol* $\perp$ *and* δ *is the transition relation.*

The transition relation $\delta \subseteq (Q \times \Sigma_{int} \times \Gamma \times Q) \cup (Q \times \Sigma_{c_1} \times \Gamma \times Q \times Push_1^{\gamma}) \cup (Q \times \Sigma_{r_1} \times \Gamma \times Q \times Pop_1) \cup (Q \times \Sigma_{c_i} \times \Gamma \times Q \times Push_i) \cup (Q \times \Sigma_{r_i} \times \Gamma \times 'N' \times Q \times Pop_i) \cup (Q \times \Sigma_{r_i} \times \Gamma \times 'Y' \times Q)$ for $\gamma \in \Gamma$ and $2 \leq i \leq n$.

Here 'Y' and 'N' are the values returned by the Is_Last function when queried with the stack and the current input alphabet. Note that the stack of a *level n* HVPA will be a *level n store* and higher order push and pop operations are possible.

Behavior of HVPA: The behavior of an HVPA on reading various inputs is given as follows:

[Push] If $a \in \Sigma_{c_1}$ then we have an ordinary push operation(*level 1* or $Push_1^{\gamma}$). If $a \in \Sigma_{c_i}$ for $2 \leq i \leq n$ then we have an *level i* or $Push_i$ push operation.

[Pop] If $a \in \Sigma_{r_1}$ then we have an ordinary pop operation(*level 1* or Pop_1). If $a \in \Sigma_{r_i}$ for $2 \leq i \leq n$ then we have an *level i* or Pop_i push operation, provided Is_Last returned 'N'.

[Internal] Internal actions are same as that of VPA.

Run of HVPA : Let M be a *level n* HVPA. For a word $w = a_1 \ldots a_k$ in Σ^*, a run of M on w is a sequence $\rho = (q_0, \sigma_0), \ldots, (q_k, \sigma_k)$, where $q_i \in Q$, σ_i is *level n*

store (stack) $q_0 \in Q_{in}, \sigma_0 = \perp$ and the sequence $\sigma_i's$ are consistent with transition relation of HVPA. For example if $a_i \in \Sigma_{r_3}$ then $q_{i+1} \in \delta(q_i, a_i)$ and σ_{i+1} is obtained from σ_i by doing a *level 3* Pop(Pop_3) operation and son on.

A run $\rho = (q_0, \sigma_0), \ldots, (q_k, \sigma_k)$ is accepting if the last state is a final state, i.e if $q_k \in Q_F$. A word $w \in \Sigma^*$ is accepted by a HVPA M if there is an accepting run of M on w. The language of M, is the set of words accepted by M. A language L is said to *higher order visibly pushdown language* (HVPL), if there is some HVPA which accepts it and we write L = L(M).

Definition (Higher Order Visibly Pushdown Languages(HVPLs)) *A language of finite words over $L \subseteq \overline{\Sigma}^*$ is a higher order visibly pushdown language (HVPL) if there is a HVPA M over $\overline{\Sigma}$ such that $L(M) = L$.*

Note: When we talk of HVPLs one thing to notice is that the partition on the input symbols is part of the definition of language itself. So the following two languages are different
$L_1 = \{a^n b^n\}$ over $\overline{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ with $\Sigma_{c_1} = \{a\}, \Sigma_{r_1} = \{b\}$ and $\Sigma_{int} = \phi$.
$L_2 = \{a^n b^n\}$ over $\overline{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ with $\Sigma_{c_2} = \{a\}, \Sigma_{r_2} = \{b\}$ and $\Sigma_{int} = \phi$.

2.4 A language accepted by HVPA but not by VPA

Consider the language L=$\{a^n b c^n d e^n | n \geq 1\}$ over $\overline{\Sigma} = \langle \Sigma_c, \Sigma_r, \Sigma_{int} \rangle$ with $\Sigma_c = \{a, b\}, \Sigma_r = \{c, d, e\}$ and $\Sigma_{int} = \phi$. It is clear that no VPA can accept (not even by PDA) can accept this language. But we can design a HVPA of *level 2* to accept this language with $\Sigma_{c_1} = \{a\}, \Sigma_{c_2} = \{b\}, \Sigma_{r_1} = \{c, e\}$, $\Sigma_{r_2} = \{d\}$ and $\Sigma_{int} = \phi$.

When it reads an a it makes an ordinary *Push* operation. When it sees b it makes a $Push_2$ operation so that now we have two copies of a^n in the stack. The topmost copy can be used for matching of c^n. When the machine encounters d it pops(Pop_2)

the top most *1-Store* only if it is empty. Now the matching of e^n can easily be done using the second copy of a^n.

We design an HVPA $M = (\{q_0, q_1, q_2, , q_{acc}\}, \{q_0\}, \{A\}, \delta, \{q_{acc}\})$ to accept L. In order to make the machine simpler, we add a marker \$ ($\$ \in \Sigma_{int}$) at the end of the input. The transitions of M are as follows:

$(q_0, a, \perp/A) \mapsto (q_0, Push_1^A)$

$(q_0, b, A) \mapsto (q_1, Push_2)$

$(q_1, c, A) \mapsto (q_1, Pop_1)$

$(q_1, d, \perp, 'N') \mapsto (q_2, Pop_2)$

$(q_2, e, A) \mapsto (q_2; Pop_1)$

$(q_2, \$, \perp) \mapsto (q_{acc})$

It is easy to see that L(M) = L.

2.5 Deterministic HVPA

Definition(Deterministic HVPA) A HVPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ is said to be deterministic if $|Q_{in}| = 1$ and for every $q, q' \in Q, \gamma, \gamma' \in \Gamma$:

- for every $a \in \Sigma_{int}$, there is at most one transition of the kind $(q, a, \gamma, q') \in \delta$
- for every $a \in \Sigma_{c_1}$, there is at most one transition of the kind $(q, a, \gamma', q', Push_1^{\gamma}) \in \delta$
- for every $a \in \Sigma_{c_i}, i > 1$, there is at most one transition of the kind $(q, a, \gamma, q', Push_i) \in \delta$
- for every $a \in \Sigma_{r_1}$, there is at most one transition of the kind $(q, a, \gamma, q', Pop_1) \in \delta$
- for every $a \in \Sigma_{r_i}, i > 1$, there is at most one transition of the kind $(q, a, \gamma, 'N', q', Pop_i) \in \delta$

• for every $a \in \Sigma_{r_i}, i > 1$, there is at most one transition of the kind $(q, a, \gamma, `Y', q') \in \delta$

2.6 Renaming Operation

A renaming of $\overline{\Sigma}$ to $\overline{\Sigma}'$ is a function $f : \Sigma \rightarrow \Sigma'$ such that $f(\Sigma_{int}) \subseteq \Sigma'_{int}$, $f(\Sigma_{c_i}) \subseteq \Sigma'_{c_i}$ and $f(\Sigma_{r_i}) \subseteq \Sigma'_{r_i}$. A renaming of f can extended to words over Σ in the natural way : $f(a_1 \ldots a_k) = f(a_1) \ldots f(a_k)$.

Chapter 3

Closure Properties of HVPLs

In this section we prove that HVPLs are closed under union, intersection, and renaming. We also show that any ordinary HPDL can be converted to a HVPL under a special kind of mapping on the set of input alphabet. We also provide counter examples to prove that HVPLs are not closed under determinization, concatenation and Kleene *.

3.1 Closure under union

Theorem 1 *(Closure under union) Let L_1 and L_2 be higher order visibly pushdown languages with respect to $\overline{\Sigma}$. Then $L_1 \cup L_2$ is higher order visibly pushdown language with respect to $\overline{\Sigma}$.*

Proof: Let $M_1 = (Q_1, Q_{in_1}, \Gamma_1, \delta_1, Q_{F_1})$ and $M_2 = (Q_2, Q_{in_2}, \Gamma_2, \delta_2, Q_{F_2})$ be HVPAs which accepts L_1 and L_2 respectively. Assume that $Q_1 \cap Q_2 = \phi$.

Define $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ such that $Q = Q_1 \cup Q_2$, $Q_{in} = Q_{in_1} \cup Q_{in_2}$, $Q_F = Q_{F_1} \cup Q_{F_2}$, $\Gamma = \Gamma_1 \cup \Gamma_2$ and $\delta = \delta_1 \cup \delta_2$.

Intuitively, M is capable of simulating all runs of both M_1 and M_2. M non-deterministically selects one of the initial states of M_1 or M_2. Suppose the selected initial state is that of M_1, then M starts to simulate a run of M_1 on the input string.

Since $Q_1 \cap Q_2 = \phi$ there is no fear of mixing up of runs of M_1 and M_2. So if the given input string w is in $L_1 \cup L_2$, then M has an accepting run on w. That implies $L_1 \cup L_2 \subseteq L$.

On the other hand, suppose M has an accepting run on w. Since the transitions of M are exactly the union of transitions of M_1 and M_2, the accepting run of M on w is one of the runs of either M_1 or M_2. So $w \in L(M) \Rightarrow w \in L_1 \cup L_2$.

Therefore $L(M) = L(M_1) \cup L(M_2)$. ♠

Alternative proof: To accept $L_1 \cup L_2$ we design a HVPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ with $Q = Q_1 \times Q_2, Q_{in} = Q_{in_1} \times Q_{in_2}, Q_F = \{Q_1 \times Q_{F_2}\} \cup \{Q_{F_1} \times Q_2\}, \Gamma = \Gamma_1 \times \Gamma_2$.

Intuitively M is trying to simulates runs of M_1 and M_2 in parallel. A run of M on the input string is accepting if one of the runs of M_1 or M_2 is accepting. i.e if M ends up in a state (q_i, q_j) on an input string w, then w is accepted if $q_i \in Q_{F_1}$ or $q_j \in Q_{F_2}$.

When reading $a \in \Sigma_{c_1}$ if M_1 pushes γ_1 and M_2 pushes γ_2 the M pushes (γ_1, γ_2). One important thing to note is that while reading a symbol $a \in \Sigma_{c_1} (a \in \Sigma_{r_1})$ the behavior of the stack of M_1 and that of M_2 is synchronized. The only difference is they *push(pop)* different symbols.

When reading $a \in \Sigma_{c_i}$ $(a \in \Sigma_{r_i})$, for $2 \leq i \leq n$ both M_1 and M_2 perform $Push_i(Pop_i)$ and that effect can be simulated by performing a $Push_i(Pop_i)$ on the stack of M. ♠

Note that the construction in the first proof gives a non-deterministic machine whereas the second proof helps us to construct a deterministic machine directly if both M_1 and M_2 are deterministic. This is important because it turns out non-deterministic HVPAs is more powerful than deterministic HVPAs (proof is given in subsequent section).

3.2 Closure under intersection

Theorem 2 *(Closure under intersection) Let L_1 and L_2 be higher order visibly pushdown languages with respect to $\overline{\Sigma}$. Then $L_1 \cap L_2$ is higher order visibly pushdown language with respect to $\overline{\Sigma}$.*

Proof: Let $M_1 = (Q_1, Q_{in_1}, \Gamma_1, \delta_1, Q_{F_1})$ and $M_2 = (Q_2, Q_{in_2}, \Gamma_2, \delta_2, Q_{F_2})$ be HVPAs which accepts L_1 and L_2 respectively.

To accept $L_1 \cap L_2$ we design a HVPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ which is almost similar to the machine we designed the alternate proof of closure under union except for the the set of final states. $Q = Q_1 \times Q_2, Q_{in} = Q_{in_1} \times Q_{in_2}, Q_F = Q_{F_1} \times Q_{F_2}, \Gamma = \Gamma_1 \times \Gamma_2$.

Here also M is trying to simulate runs of M_1 and M_2 in parallel. M accepts a run only if both M_1 and M_2 accepts their respective simulated run. i.e if M ends up in a state (q_i, q_j) on an input string w, then w is accepted if and only if $q_i \in Q_{F_1}$ and $q_j \in Q_{F_2}$.

When reading $a \in \Sigma_{c_1}$ if M_1 pushes γ_1 and M_2 pushes γ_2 the M pushes (γ_1, γ_2). Similarly the higher order push and pop operations of M_1 and M_2 can be simulated on the stack of M.

It is is to verify that that $L(M) = L_1 \cap L_2$ ♠

3.3 Closure under renaming

Theorem 3 *(Closure under renaming) Let L be higher order visibly pushdown languages with respect to $\overline{\Sigma}$ and if f is a renaming of $\overline{\Sigma}$ to $\overline{\Sigma}'$, then $f(L)$ is a higher order visibly pushdown language with respect to $\overline{\Sigma}'$.*

Proof: Let $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ be HVPA accepting L. We design a HVPA $M' = (Q, Q_{in}, \Gamma, \delta', Q_F)$ over $\overline{\Sigma}'$ which accepts $f(L)$.

Transform each transition of M on a to a transition on $f(a)$. For example, if $(q_i, a, \gamma, q_j, op) \in \delta$, then $(q_i, f(a), \gamma, q_j, op) \in \delta'$. Note that restriction that f maps calls of *level i* to calls *level i* and returns of *level i* to returns of *level i* is necessary for HVPLs to be closed under renaming operations. By the definition of δ' itself it clear that $w \in L \Leftrightarrow f(w) \in f(L)$ and hence the theorem. ♠

3.4 Conversion of HPDA to HVPA under a special mapping on input alphabet

Theorem 4 *Let L' be a level n language (i.e there is a level n HPDA accepts L') over Σ'. Then there is a level n higher order visibly pushdown language L over $\overline{\Sigma}$ with $\Sigma_{c_i} = \Sigma' \times \{c_i\}$, $\Sigma_{r_i} = \Sigma' \times \{r_i\}$ and $\Sigma_{int} = \Sigma' \times \{int\}$ for $1 \leq i \leq n$, such that $f(L) = L'$.*

Proof: Let $M' = (Q, Q_{in}, \Gamma, \delta', Q_F)$ be a *level n* HPDA accepts L'. We design a HVPA $M = (Q, Q_{in}, \Gamma, \delta, Q_F)$ of *level n* to accept to L. This can be done by restricting the transitions according to the input read. For example if we have a $Push_i$ from state q_i to q_j for M' by reading an alphabet a then M will have transition from from state q_i to q_j on reading alphabet (a, c_i) and so on.

We define the transition relation δ as follows. Note that $\delta' \subseteq (Q \times \Sigma \times \Gamma \times Q \times Op_n)$ where $Op_n = \{push_1^{\gamma}\} \cup \{push_m\} \cup \{pop_k\}$, for $2 \leq m \leq n$, $\gamma \in \Gamma$, and $1 \leq k \leq n$. If $(q_i, a, \gamma', q_j, op) \in \delta$ for $\gamma' \in \Gamma$ then,

- **Case 1:** $op = push_1^{\gamma}$ for some $\gamma \in \Gamma$, then $(q_i, (a, c_1), \gamma', q_j, push_1^{\gamma}) \in \delta$.
- **Case 2:** $op = push_m$ for $2 \leq m \leq n$, then $(q_i, (a, c_m), \gamma', q_j, push_m) \in \delta$.
- **Case 3:** $op = pop_k$ for $1 \leq k \leq n$, then $(q_i, (a, r_k), \gamma', q_j, pop_k) \in \delta$.

In our definition of HPDA, every transition changes the stack contents. If we allow transitions which does not change the stack contents then they will appear as the internal operations in δ.

Note that elements of Σ are pairs. Now we define f to be the projection operation which takes elements of Σ and simply projects the first component of the pair. Now we argue that $f(L) = L'$.

Suppose $w' \in L'$, then M' has an accepting run on w'. We will create a decorated string w from w' as follows. Let the transition made on $a_i (1 \leq i \leq |w'|)$ be $(q_i, a, \gamma, q_j, op)$, then the corresponding letter in w be of the form (a, x) where $x \in \{c_1, \ldots c_n, r_1, \ldots, r_n, int\}$ and x is determined by what action is done on the stack. For example, if the action on a_i is Pop_m then (a_i, r_m) would be the corresponding letter in w. By definition of δ, there is a Pop_m operation on (a_i, r_m). Since M' has an accepting run on w', it is obvious that M also has an accepting run w.

Suppose $w \in L$. Let w' is obtained by taking the first component of letters in w. Again by the way δ defined, for every the transitions in the accepting run of M on w, the same operation is possible on M' also, which makes M' to accept w'. ♠

3.5 HVPLs are not closed under determinization

In this section we show that nondeterministic HVPA (NHVPA) is more powerful than a deterministic one (DHVPA). In other words, there are languages that can be accepted by an NHVPA that cannot be accepted by a DHVPA.

Consider the following language :

L = $\{w_1 w_2 c {w_2}^R \overline{w_1}^R d a_1^n b_1^m$: $n = |w_2|$, $m = |w_1|$ and $n, m \geq 1\}$ where w_1,$w_2 \in \{a, b\}^*$, ${w_2}^R$ is the reverse of w_2 with a and b replaced by a_1 and b_1 respectively and $\overline{w_1}^R$ is the reverse of w_1 with b and a replaced by a_1 and b_1 respectively.

For example: If $w_1 = abaa$ and $w_2 = bbab$ then, $\overline{w_1}^R = b_1b_1a_1b_1$ and $w_2{}^R = b_1a_1b_1b_1$.

The partition on $\overline{\Sigma}$ is as follows : $\Sigma_{c_1} = \{a, b\}, \Sigma_{c_2} = \{c\}, \Sigma_{r_1} = \{a_1, b_1\}, \Sigma_{r_2} = \{d\}$ and $\Sigma_{int} = \phi$.

We design an NHVPA

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_{acc}\}, \{q_0\}, \{A, B, A_1, B_1\}, \{a, b, a_1, b_1\}, \delta, \{q_{acc}\})$ to accept L. In order to make the machine simpler, we add a marker \$ (\$$\in \Sigma_{int}$) at the end of the input. The transitions of M are as follows:

$(q_0, a, \perp) \mapsto (q_1/q_2, Push_1^A)$
$(q_0, b, \perp) \mapsto (q_1/q_2, Push_1^B)$
$(q_1, a, A/B) \mapsto (q_1/q_2, Push_1^A)$
$(q_1, b, A/B) \mapsto (q_1/q_2, Push_1^B)$
$(q_2, a, A/B/A_1/B_1) \mapsto (q_2, Push_1^{A_1})$
$(q_2, b, A/B/A_1/B_1) \mapsto (q_2, Push_1^{B_1})$
$(q_2, c, A_1/B_1) \mapsto (q_3, Push_2)$
$(q_3, a_1, A_1) \mapsto (q_3, Pop_1)$
$(q_3, b_1, B_1) \mapsto (q_3, Pop_1)$
$(q_3, a_1, B) \mapsto (q_4, Pop_1)$
$(q_3, b_1, A) \mapsto (q_4, Pop_1)$
$(q_4, a_1, B) \mapsto (q_4, Pop_1)$
$(q_4, b_1, A) \mapsto (q_4, Pop_1)$
$(q_4, d, \perp, 'N') \mapsto (q_5, Pop_2)$
$(q_5, a_1, A_1) \mapsto (q_5, Pop_1)$
$(q_5, a_1, B_1) \mapsto (q_5, Pop_1)$
$(q_5, b_1, A) \mapsto (q_6, Pop_1)$
$(q_5, b_1, B) \mapsto (q_6, Pop_1)$
$(q_6, b_1, A) \mapsto (q_6, Pop_1)$
$(q_6, b_1, B) \mapsto (q_6, Pop_1)$
$(q_6, \$, \perp) \mapsto (q_{acc})$

Intuitively M guesses the split between w_1 and w_2, and marks it in the stack by using different set of stack alphabets for w_1 and w_2 (A and B for w_1 and A_1 and B_1 for w_2). When M encounters c, the stack is replicated. So while reading ${w_2}^R\overline{w_1}^R$, M can verify whether the split made is correct or not by using the top copy of the stack. If the split happens to be correct, then M has to verify whether $a_1^n b_1^m$ with $n = |w_2|$ and $m = |w_1|$ holds or not. This can be done using the bottom copy of the stack.

It is easy to see that L(M) = L.

Now we will prove that L is not accepted by any DHVPA.

Theorem 5 *L (as defined above) can not be accepted by a DHVPA.*

Proof: The proof is by contradiction. Assume that there is a DHVPA M_1 such that L(M_1) = L. Let p be the number states of M_1.

Consider an input string w such that $|w| > p + 1$. Observe that the number of splits possible on w into w_1 and w_2 is $|w| - 1$, which is $> p$. When M_1 completes reading $w_1 w_2 c$, the stack of M_1 will have two copies same *1-store*. After reading $w_1 w_2 c {w_2}^R \overline{w_1}^R d$, the topmost *1-store* would have got popped and M_1 will be in some state say q. Note that the split is decided uniquely by the substring ${w_2}^R\overline{w_1}^R$ and there can be more than p possible splits. Therefore for some of the splits the automaton will have to end up in same state with same stack contents. For example,

Let $u_1u_2 = v_1v_2$ such that $|u_1| \neq |v_1|$, and for both $u_1u_2c{u_2}^R\overline{u_1}^R d$ and $v_1v_2c{v_2}^R\overline{v_1}^R d$, M_1 ends up in the same state, say q. Since both $u_1u_2c{u_2}^R\overline{u_1}^R d{a_1}^{|u_2|}{b_1}^{|u_1|}$ and $v_1v_2c{v_2}^R\overline{v_1}^R d{a_1}^{|v_2|}{b_1}^{|v_1|}$ are in L, M_1 can reach an accepting state from q on both ${a_1}^{|u_2|}{b_1}^{|u_1|}$ and ${a_1}^{|v_2|}{b_1}^{|v_1|}$. This would make M_1 to accept $u_1u_2c{u_2}^R\overline{u_1}^R d{a_1}^{|v_2|}{b_1}^{|v_2|}$, a string which is obviously not in L. So we arrive at a contradiction to the assumption that L(M_1)=L. ♠

So we have got a language L accepted by NHVPA and not by any DHVP, which proves the following theorem.

Theorem 6 *HVPLs are not closed under determinization.*

It is interesting to observe that even though L is not determizable, it can be complemented. That is, we can design *2-level* HVPA to accept $\overline{L}$. There are 3 cases which makes a string w to be in $\overline{L}$.

- **Case 1:** w is ill formed, i.e either w is not in the form $ucvda_1^n b_1^m$ with $u \in \{a,b\}^*$ and $v \in \{a_1,b_1\}^*$ or even if $w = ucvda_1^n b_1^m$, the sizes of the substrings do not agree. That is, either $|u| \neq v|$ or $|u| \neq n+m$.

- **Case 2:** w is in the correct format but there are no w_1 and w_2 such that w is of the form $w_1 w_2 c {w_2}^R \overline{w_1}^R d a_1^n b_1^m$.

- **Case 3:** w is of the form $w_1 w_2 c {w_2}^R \overline{w_1}^R d a_1^n b_1^m$, with $n \neq |w_2|$.

Designing deterministic HVPAs to check whether w satisfies case 1 or case 2 is easy. Let M_1 and M_2 be HVPAs which checks conditions 1 and 2 respectively. For case 3, we can design a non-deterministic M_3 machine similar to one which we designed to accept L. The only difference M_1 has with M is that, when M_1 detects that the number of a_1's is not equal to $|w_2|$, then M_1 should indicate it by going to an accepting state and staying there till the entire input has been read.

If a string $w \in \overline{L}$, then it comes from on of the above three cases. So $L(M_1) \cup L(M_2) \cup L(M_3) = \overline{L}$ is trivial. Now we design an HVPA M' which accept $L(M_1) \cup L(M_2) \cup L(M_3)$.

3.6 Some interesting *level* − 2 HVPLs

L_1 and L_2 over $\overline{\Sigma}$ under the partition $\Sigma_{c_1} = \{a,b\}, \Sigma_{c_2} \doteq \{c\}, \Sigma_{r_1} = \{a_1,b_1\}$, $\Sigma_{r_2} = \{d\}$ and $\Sigma_{int} = \phi$ are defined as follows:

$$L_1 = \{ w_1 x w_2 \; c \; {w_2^{netR}} {w_1}^{netR} \; d \; a_1^{|w_2^{netR}|} b_1^{|w_1^{netR}|} \}$$
$$L_2 = \{ w_1 x w_2 \; c \; {w_2^{netR}} \overline{w}_1{}^{netR} \; d \; a_1^{|w_2^{netR}|} b_1^{|w_1^{netR}|} \}$$

where w_1,w_2,$x \in \{a, b, a_1, b_1\}^*$, ${w_2}^{netR}$ (${w_1}^{netR}$) is the *"net reverse"* of w_2 (w_1) with a and b replaced by a_1 and b_1 respectively, ${\overline{w}_1}^{netR}$ is the *"net reverse"* of w_1 with a and b replaced by b_1 and a_1 respectively, x is a string whose no prefix has more number of return symbols than call symbols and the stack content is the same before and after reading x. *"net reverse"* of a string is the reverse of the subsequence of unreturned calls. For example, if $w_1 = abbaa_1b_1bab_1$, then the subsequence of unreturned calls is, abb. Then ${w_1}^{netR}$ is $b_1b_1a_1$ and ${\overline{w}_1}^{netR}$ is $a_1a_1b_1$.

Observation 1 L_1 is accepted by DHVPA.

In a first look it may appear that the split into w_1, x and w_2 is important. But if we observe a bit closer it becomes clear that the split is not important. Because in whichever split we take the stack content after reading w_1xw_2 would be the same. So matching of $w_2^{netR}{w_1}^{netR}$ becomes easy. In other words L_1 and the following language are one and the same.

$L_3 = \{w\ c\ w^{netR}\ d\ a_1^n b_1^m : |w| = n + m\}$

Observation 2 L_2 is accepted by NHVPA but not by DHVPA.

By using similar arguments we used to show that L(as defined in the proof to show that HVPLs are not determinizable) is not determizable, we can show that L_2 is also not determinizable. In case of L_2 also we can design an NHVPA such that if a string is in the language, then there is unique accepting run. Again by similar techniques we used to complement L, we can complement L_2 also.

We give a HVPL L_4,whose complement $\overline{L}_4$ is believed not to be an HVPL. L_4 over $\overline{\Sigma}$ under the partition $\Sigma_{c_1} = \{a, b\}, \Sigma_{c_2} = \{c\}, \Sigma_{r_1} = \{a_1, b_1\}$, $\Sigma_{r_2} = \{d\}$ and $\Sigma_{int} = \phi$ is defined as follows:

$L_4 = \{w_1xw_2\ c\ w_2^R y{\overline{w}_1}^R\ d\ a_1^n \overline{x}^R b_1^m : n = |w_2|\ ,\ m = |w_1| \text{and } |x| = |y|\}$

where w_1,w_2,$x \in \{a,b\}^*$, ${w_2}^R$ is the reverse of w_2 with a and b replaced by a_1 and b_1 respectively and $\overline{w_1}^R$ is the reverse of w_1 with b and a replaced by a_1 and b_1 respectively. $y \in \{a_1,b_1\}^*$, $\overline{x}^R$ is the reverse of x with b and a replaced by a_1 and b_1 respectively.

Observation 3 L_4 is accepted by NHVPA but not by DHVPA.

We design a non-deterministic HVPA, M_4 that accepts L_4. Intuitively M_4 guesses the splits w_1, x and w_2 and marks them on the stack using different set of stack alphabets and while reading ${w_2}^R y {\overline{w}_1}^R \ d\ a_1^n \overline{x}^R b_1^m$, M_4 can verify whether the guessed split was correct or not. M_4 accepts the input string if the guess was correct. We use A and B for w_1, A_x and B_x for x and A_1 and B_1 for w_2 as the stack alphabets. In order to make the machine simpler, we add a marker \$ (\$$\in \Sigma_{int}$) at the end of the input. M_4 is defined formally as follows:

$$M_4 = (\{q_0 \ldots q_{10}, q_{acc}\}, \{q_0\}, \{A, B, A_x, B_x, A_1, B_1\}, \{a, b, a_1, b_1\}, \delta, \{q_{acc}\})$$

The transitions of M_4 are as follows:

$(q_0, a, \perp/A/B) \mapsto (q_0/q_1/q_2, Push_1^A)$
$(q_0, b, \perp/A/B) \mapsto (q_0/q_1/q_2, Push_1^B)$
$(q_1, a, A/B/A_x/B_x) \mapsto (q_1/q_2, Push_1^{A_x})$
$(q_1, b, A/B/A_x/B_x) \mapsto (q_1/q_2, Push_1^{B_x})$
$(q_2, a, A/B/A_x/B_x) \mapsto (q_3, Push_1^{A_1})$
$(q_2, b, A/B/A_x/B_x) \mapsto (q_3, Push_1^{B_1})$
$(q_3, a, A_1/B_1) \mapsto (q_3, Push_1^{A_1})$
$(q_3, b, A_1/B_1) \mapsto (q_3, Push_1^{B_1})$
$(q_4, c, A_1/B_1) \mapsto (q_5, Push_2)$
$(q_5, a_1, A_1) \mapsto (q_5, Pop_1)$
$(q_5, b_1, B_1) \mapsto (q_5, Pop_1)$
$(q_5, a_1, A_x/B_x) \mapsto (q_6, Pop_1)$
$(q_5, b_1, A_x/B_x) \mapsto (q_6, Pop_1)$
$(q_6, a_1, A_x/B_x) \mapsto (q_6, Pop_1)$

$(q_6, b_1, A_x/B_x) \mapsto (q_6, Pop_1)$
$(q_6, a_1, B_1) \mapsto (q_7, Pop_1)$
$(q_6, b_1, A_1) \mapsto (q_7, Pop_1)$
$(q_7, a_1, B_1) \mapsto (q_7, Pop_1)$
$(q_7, b_1, A_1) \mapsto (q_7, Pop_1)$
$(q_7, d, \perp, `N') \mapsto (q_8, Pop_2)$
$(q_8, a_1, A_1) \mapsto (q_8, Pop_1)$
$(q_8, a_1, B_1) \mapsto (q_8, Pop_1)$
$(q_8, a_1, B_x) \mapsto (q_9, Pop_1)$
$(q_8, b_1, A_x) \mapsto (q_9, Pop_1)$
$(q_9, a_1, B_x) \mapsto (q_9, Pop_1)$
$(q_9, b_1, A_x) \mapsto (q_9, Pop_1)$
$(q_9, b_1, A/B) \mapsto (q_{10}, Pop_1)$
$(q_{10}, b_1, A/B) \mapsto (q_{10}, Pop_1)$
$(q_{10}, \$, \perp) \mapsto (q_{acc})$

Now we will give some intuitions for why $\overline{L}_4$ does not seem to be a HVPL. Unlike from the other languages given above it seems that no HVPA can be defined such that the accepting run of an input string is unique. In other languages it was easy because the split in to w_1 and w_2 was uniquely determined by the substring between c and d. But in case of L_4 the split is determined by the substring after d. Moreover in case of other languages there was a unique accepting run for strings in the language. But in case of L_4 the number of accepting runs is unbounded. That means, for large enough strings there can be many accepting runs and the number accepting runs can not bounded by any number. So in order to accept $\overline{L}_4$, a HVPA should be able to decide whether a given string is not accepted in any of its runs, which seems to be an impossible task. Unfortunately we do not have a formal proof for that.

3.7 HVPLs are not closed under Concatenation and Kleene *

It turns out that HVPLs are not closed under concatenation. In order to show an HVPL, L is closed are closed under concatenation , we should be able to simulate two strings in L without mixing up the stacks of different simulation. The example given below shows that it is impossible to have a such a simulation without mixing up of stacks.

Theorem 7 *HVPLs are not closed under concatenation.*

Proof: We give an HVPL, L and show that L.L is not HVPL. L is defined as follows:

L = $\{a^n b^m : n \geq m, n > 0\}$ over $\overline{\Sigma}$ with partion : $\Sigma_{c_1} = \phi, \Sigma_{c_2} = \{a\}, \Sigma_{r_1} = \phi, \Sigma_{r_2} = \{b\}$ and $\Sigma_{int} = \phi$.

It is easy to see that the HVPA given below accepts L.

$M' = (\{q_0, q_{acc}, q_{rej}\}, \{q_0\}, \{\}, \{a, b\}, \delta, \{q_{acc}\})$

The transitions of M' are as follows:

$(q_0, a, \perp) \mapsto (q_{acc}, Push_2)$

$(q_{acc}, a, \perp) \mapsto (q_{acc}, Push_2)$

$(q_{acc}, b, \perp,' N') \mapsto (q_{acc}, Pop_2)$

$(q_{acc}, b, \perp,' Y') \mapsto (q_{rej})$

Now we will prove that no HVPA accepts L.L. Proof is again by contradiction.

Assume that there is an HVPA M such that L(M) = L.L. Let p be the number states of M. Consider an input string $w = a^n b^n$ such that $n > p$. While reading b's, since $n > p$ some of the states of M should repeat. Let q be one of the repeated state in the simulation of $w = a^n b^n$. Let the number of stacks popped between two

q's be k. That means in an accepting run the state q is repeated and in between the repetition we could pop k number of stacks. So we can repeat the moves from q to q by popping k more stacks and still end up in an accepting state. Now consider the run of M on the input string $a^{(i+k)}b^i.a^nb^{n+k}$. After reading $a^{(i+k)}b^i$, the stack of M will have k empty *1-store*. This makes M to accept $a^{(i+k)}b^i.a^nb^{n+k}$ because we could have repeated the moves between two q's by popping k more stacks and still ending up in a final state. Since $a^{(i+k)}b^i.a^nb^{n+k}$ is not in L.L. we arrive at a contradiction to the assumption that M accepts L.L. ♠

Theorem 8 *HVPLs are not closed under Kleene *.*

Proof: Consider the language L defined in the proof of the above theorem. We show that no HVPA accepts L^*. Note that $L^* := L^0 \cup L^1 \cup \ldots$.

Assume that there is an HVPA M such that L(M) = L^*. Since M accepts L^*, it should accept all strings in L^2. Then using the same arguments we used in the proof of the above theorem we can show that M accepts strings of the form $a^{(i+k)}b^i.a^nb^{n+k}$ which are obviously not in L^2. Moreover it is easy to see that strings of the form $a^{(i+k)}b^i.a^nb^{n+k}$ does not belong to any $L^j, j \neq 2$. So we arrive at contradiction to the assumption that L(M) = L^*. ♠

3.8 Difficulties faced in giving a logical characterization for HVPLs

In [1] a logical characterization for VPLs was given in terms of Monadic second order logic (MSO_μ). Fix $\overline{\Sigma}$. A word w over Σ can be treated as a structure over the universe $U = 1, \ldots, |w|$ that denotes the set of positions and a set of unary predicates Q_a, for each $a \in \Sigma$, where $Q_a(i)$ is true iff $w[i] = a$. The "match" relation of calls and returns μ over U is defined as follows : $\mu(i, j)$ is true iff $w[i]$ is a call and $w[j]$ is the matching return. Let $x, y, \ldots$ denote the first-order variables and $X, Y, \ldots$ denote the second-order(set) variables. The MSO_μ over $\overline{\Sigma}$ was defined as :

$\phi = Q_a(x) \mid x \in X \mid x \leq y \mid \mu(x,y) \mid \neg\phi \mid \phi \vee \phi \mid \exists x.\phi \mid \exists X.\phi$
where $a \in \Sigma$, x is a first-order variable and X is a set variable.

Note that in case of VPLs, there is an inherent matching in the run of a VPA on a word w. i.e For any $a \in \Sigma_c$ that appears in w, either it is a unreturned call or the symbol pushed by the VPA while reading a will be popped by some $b \in \Sigma_r$. This way we can associate every call(except for unreturned calls) with exactly one return that appears in later part of w. But such a matching is not so clear in case of HVPAs, because we could pop an entire stack by a higher order pop operation. So definition of the "match" relation μ became a tough task.

One possible way of defining the "match" relation for an *n-level*-HVPA could be defining a set of predicates $\mu_i's, 1 \leq i \leq n$ in the following manner. Let w be a word over $\overline{\Sigma}$. $\mu_1(i,j) = true$ if $w[i] \in \Sigma_{c_1}$, $w[j] \in \Sigma_{r_1}$ and when reading $w[j]$, the symbol popped is the symbol pushed when $w[i]$ was read. Note that μ_1 can be a many-one relation because of the stack replication. That is, the symbol pushed by $w[i] \in \Sigma_{c_1}$ could be popped by many $w[j]'s \in \Sigma_{r_1}$ if the stack after reading $w[i]$ was replicated. $\mu_i(i,j) = true$, for $i > 1$ if $w[i] \in \Sigma_{c_i}$, $w[j] \in \Sigma_{r_i}$ and when reading $w[j]$, the stack popped is the stack replicated when $w[i]$ was read.

We believe that it could be possible that the runs of HVPAs can be characterized by the boolean closure of the set of logical relations $\mu_i's$. In order to make things clear further investigation has to be done.

Chapter 4

Higher order Visibly Pushdown ω-Languages

In early sixties, Richard Buchi introduced the concept of finite automata on infinite words in [10]. Working on weak theories of integers, he was lead to consider the monadic second order theory of the successor function on integers. He was able to prove the decidability of this theory. Later on, in [11] Robert McNauhton proved the equivalence of deterministic and non-deterministic automata, a natural extension of the corresponding result for finite words. This difficult result was conjectured by David Muller while working at questions related to oscillating circuits. The notion of an infinite sequence is of interest to model the behavior of systems which are supposed to work endless, as operating systems for example.

In this section we introduce HVPA on infinite words ω-HVPA and shows they are closed under union, intersection and renaming. We also show that non-deterministic Buchi and Muller ω-HVPA can simulate each other.

Definition(Higher order Visibly Pushdown ω-Automata) *An ω-HVPA is a tuple $M = (Q, Q_{in}, \Gamma, \delta, \boldsymbol{F})$ where Q, Q_{in}, Γ and δ are same as that in HVPA and $\boldsymbol{F}$ is the "acceptance component" of the ω-HVPA.*

For any $\alpha \in \Sigma^{\omega}$, a run is a sequence $\rho = (q_0, \sigma_0), (q_2, \sigma_2) \ldots$ is a natural extension of the definition of run over finite words. Contrary to HVPA on finite machines, ω-HVPA runs for ever. While the acceptance condition of automata on finite words is rather canonical, there are many possibilities of defining acceptance on infinite words. An acceptance condition restricts the occurrence of states in a run ρ under consideration.

We define $inf(\rho) \subseteq Q$ as follows:

we write $\exists^{\omega}$ for the quantifier "there exists infinitely many", then

$inf(\rho) = q \in Q : \exists^{\omega} i \ \rho(i) = q$ which is the set of all states that occur in ρ infinitely often.

There are two widely used acceptance conditions namely Buchi and Muller acceptance conditions:

* Buchi acceptance condition : $\mathbf{F} = F \subseteq Q$ is a set of states; a run ρ is accepting if F is met infinitely often along the run. i.e $inf(\rho) \neq \phi$.

* Muller acceptance condition : $\mathbf{F} = F = \{F_1, \ldots, F_k\}$ where each $F_i \subseteq Q$; a run ρ is accepting if the set of states it meets infinitely often is an element of $\mathbf{F}$. i.e $inf(\rho) \in \mathbf{F}$.

An infinite word α is accepted by M if there is some accepting run of M on α. The language of M, $L_{\omega}(M)$, is the set of all ω-words that it accepts. A language of infinite words $L \subseteq \Sigma^{\omega}$ is said to be an ω-HVPL if there is some ω-HVPA M such that $L = L_{\omega}(M)$.

4.1 Equivalence of Non deterministic Buchi and Muller ω-HVPAs

Theorem 9 *Non deterministic Buchi and Muller ω-HVPA can simulate each other.*

Proof : In [12] it is shown that in case of finite automata the above theorem holds. We also adopt a similar proof.

A Buchi ω-HVPA can easily be simulated by a Muller ω-HVPA, by collecting, in its acceptance component $\mathbf{F}$, those states which lead to acceptance in the given Buchi ω-HVPA.

In turn given a Muller ω-HVPA with acceptance component $\mathbf{F}$, a corresponding Buchi ω-HVPA guesses in advance the set $F \in \mathbf{F}$ of states to be visited infinitely often, and also guesses the point on its input α from which onwards only states in F will be seen. From there onwards it suffices to check that the visited states fill the set F again and again. This can be signaled by the Buchi acceptance condition. ♠

4.2 Closure properties of ω-HVPLs

By using similar techniques we used for HVPLs the last section, it is easy to see that the following theorem holds. Note that the notion of renaming can be extended to infinite words in a natural way as follows :
$f(a_1a_2\ldots) = f(a_1)f(a_2)\ldots$

Theorem 10 ***(Closure)*** *Let L_1 and L_2 be higher order ω-visibly pushdown languages with respect to $\overline{\Sigma}$. Then $L_1 \cup L_2$ and $L_1 \cap L_2$ are higher order ω-visibly pushdown languages with respect to $\overline{\Sigma}$. If f is a renaming of $\overline{\Sigma}$ to $\overline{\Sigma}'$, the f(L) is higher order ω-visibly pushdown language with respect to $\overline{\Sigma}'$.*

4.3 Deterministic Vs Non-Deterministic ω-HVPLs

Theorem 11 ***(Deterministic Vs Non-Deterministic ω-HVPLs)*** *ω-HVPL are not determinizable.*

Proof: Consider the following language L_{repbdd} :

L_{repbdd} consists of all words $\alpha \in \{c, r\}^{\omega}$, (where c is a call and and r is a return), that is repeatedly bounded- i.e. $\alpha \in L_{repbdd}$ if there is some $n \in N$ such that the stack depth on reading α infinitely often is is less than or equal to n. This kind of acceptance condition was first considered in [13].

In [1] it has been showed that L_{repbdd} can be accepted by a $1 - level$ non-deterministic Buchi ω-VPA and can not be accepted by any $1 - level$ deterministic Muller ω-VPA. ♠

Chapter 5

Conclusion and future work

In this work we have defined HVPAs, and the closure properties of the languages accepted by them are studied. We also have shown the equivalence Buchi and Muller acceptance conditions for ω-HVPLs . We strongly believe that HVPLs are not complementable, but a formal proof is yet to be obtained. In [1] it is has been proved that even though ω-VPLs are not determinizable, they are complementable. But it is not known that whether ω-HVPLs are complementable or not. We expect that there is room for further research in this direction.

Another direction that can be explored would be studying about the decision procedures and its complexity for HVPLs. In [14] it has been shown that the the emptiness problem for HPDAs (both deterministic and non-deterministic) are non-elementary complete. Since HVPAs are a restrictive version of ordinary HPDAs, it would be interesting to investigate for a procedure of elementary complexity.

Recently in [5] games on VPAs are defined and its properties have been studied. They have established that, unlike pushdown games with pushdown winning conditions, visibly pushdown games are decidable and are 2-EXPTIME-complete. Since HVPLs are closed under intersection, one may explore this direction by looking at the games defined on HVPAs and try to see for what kind of winning conditions they are decidable. In [5] it has been further proved that the topological complexity of visibly pushdown languages is a subclass of boolean combinations of Σ_3 sets. It

would be interesting to see the topological complexity of HVPLs and position of it in the Borel Hierarchy.

For the research community it has a been a greater interest to establish the the relation between various kinds of automata and the grammar counter part of it. Grammar characterizations for higher order PDA are given in [7] and [8] and known as indexed grammars. One may try to see the characterization of HVPLs in terms of the grammar.

A lot of variations of different kinds of automata, and their power in terms of computation also been studied for a long time. Alternating automata, Tree automata, Automata acting on trees, Infinite automata, Automata on infinite objects etc are some of the mostly investigated ones. But from the literature it seems that not much research has been done with higher order pushdown automata in this direction.

Bibliography

[1] R Alur and P Madhusudan. *Visibly Pushdown Automata.* In *Proceedings of the 36th annual ACM symposium on theory of computing,STOC'04*, 2004.

[2] R Alur, K Etessami and P Madhusudan. *A temperal logic of nested calls and returns.* In *TACAS*, 2004. pages 467-481

[3] Pitcher C. *Visibly pushdown effects for XML stream processing.* In *PLAN-X*, 2005.

[4] A Murawski and I Walukiewics. *Third order idealized algol with iteration is decidable.* In *FOSSACS*, 2004.

[5] R C Loding, P Madhusudan and O Serre . *Visibly Pushdown Games.* In *Proceedings of FSTTCS'04. LNCS*, 2004.

[6] R A.V Aho. *Nested stack automata.* In *JACM 16*, 1969. pages 383-406.

[7] A N Maslove. *The heirarchy of indexed languages of an arbitrary level.* In *Soviet Math. Dokl. 15* , 1974. pages 1170-1174

[8] A N Maslove. *Multi level stack automata.* In *Probl. of Inf. Transm* , 1976. pages 38-43

[9] W Damm and A Geordt. *An automata-theoretic charecterization of the OI-heirarchy.* In *Proceedings of 9th ICALP, Aarhus* , 1982. pages 141-153.

[10] J.R Buchi. *On a decision method in restricted second-order arithmetic.* In *Proceedings of International Congress for Logic, Methodology and Philosophy of science*, Stanford University Press, Stanford, 1962. pages 1-11.

[11] R McNaughton. *Testing and generating infinite sequences by a finite automaton.* In *Information and Control, 9(5)*, 1966. pages 521-530.

[12] W Thomas. *Automata on infinite objects.* In J. van Leeuwen, editor, *Handbook of Theorectical Computer Science*, volume B, pages 133-191. Elsevier Science Publishers, 1990.

[13] J Cachat, J Duprac and W.Thomas. *Solving games with Σ_3 winning condition.* In *In proceedings of 1th annual conference of the European Association for Computer Science Logic, LNCS volume 2471*, 2002. pages 322-336.

[14] J Engelfreit. *Iterated pushdown automata and coplexity classes.* In *In proceedings of 15th annual ACM symposium on theory of computing* , 1983. pages 365-373.